DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Comparison of selected model order reduction methods

Studienarbeit (T3_3101)

im Rahmen der Prüfung zum

Bachelor of Science (B.Sc.)

des Studienganges Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Florian Braun

Januar 2018

-Sperrvermerk-

Abgabedatum:	22. Mai 2023
Bearbeitungszeitraum:	14.10.2022 - 22.05.2023
Matrikelnummer, Kurs:	7433149, TINF20B1
Gutachter der Dualen Hochschule:	Lutz Gröll

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit (T3_3101) mit dem Thema:

Comparison of selected model order reduction methods

gemäß § 5 der "Studien- und Prüfungsordnung DHBW Technik" vom 29. September 2017 selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den March 15, 2023

Nachname, Vorname

Abstract

- Deutsch -

Dies ist der Beginn des Abstracts. Für die finale Bachelorarbeit musst du ein Abstract in deinem Dokument mit einbauen. So, schreibe es am besten jetzt in Deutsch und Englisch. Das Abstract ist eine kurze Zusammenfassung mit ca. 200 bis 250 Wörtern.

Versuche in das Abstract folgende Punkte aufzunehmen: Fragestellung der Arbeit, methodische Vorgehensweise oder die Hauptergebnisse deiner Arbeit.

Contents

List of abbreviations

DE	Differential Equation
DFT	Discrete Fourier Transform
FEM	Finite Element Method
FT	Fourier Transform
FFT	Fast Fourier Transform
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
LS	Least Squares
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation

List of Figures

List of Tables

source-codeverzeichnis

1. Fundamentals

1.1. Heat Equation

The conduction of heat within a medium can be described using the following partial differential equation (PDE):

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u \tag{1.1}$$

With u being a function of space and time and α being a positive constant. For this paper u will be defined in terms of one spacial dimension:

$$u := u(x,t) \tag{1.2}$$
$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \tag{1.3}$$
$$x \in \chi \subset \mathbb{R} \quad t \in \tau \subset \mathbb{R} \tag{1.4}$$
$$x_0 \leq x \leq x_n \quad t_0 \leq t \leq t_n \tag{1.5}$$

[1]
In order to not only model the conduction of heat within a medium but also a heating process, a new function $h : \chi \times \tau \to \mathbb{R}$ is introduced:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + h(x,t) \tag{1.6}$$

For this paper it is assumed that the initial condition is known:

$$f : \chi \to \mathbb{R} \tag{1.7}$$
$$u(x,t_0) = f(x) \tag{1.8}$$

1.1.1. Solving the Heat Equation using Fourier transform

Applying the Fourier transform (FT) w.r.t x to 1.6 yields the inhomogeneous ordinary differential equation (ODE):

$$\hat{u} = \mathfrak{F}(u) \quad \hat{h} = \mathfrak{F}(h) \tag{1.9}$$
$$\frac{d}{dt}\hat{u} = -\alpha\omega^2\hat{u} + \hat{h} \tag{1.10}$$

A solution to 1.10 is given by:

$$\hat{u} = \hat{u}_0 + \hat{u}_p \tag{1.11}$$

Where $\hat{u}_0$ is the homogeneous solution and $\hat{u}_p$ is the particular integral. In order to solve this ODE for the particular integral $\hat{h}$ has to be known. [2] The choice of h is, except to some restrictions, arbitrary. Therefore an approximate solution to 1.10 $\hat{u}_a$ is obtained by the forward euler scheme:

$$\frac{d}{dt}\hat{u} \approx \frac{\Delta\hat{u}}{\Delta t} \tag{1.12}$$

$$\hat{u}_{t+1} = \hat{u}_t + \Delta t(-\alpha\omega^2\hat{u} + \hat{h}) \tag{1.13}$$

$$\hat{u}_a = [\hat{u}_{t_0}, ..., \hat{u}_{t_n}] \tag{1.14}$$

In order to apply the euler scheme successfully an initial condition $\hat{u}_0$ has to be known. This initial condition is obtained by applying the discrete Fourier transform (DFT) to an initial temperature distribution along x:

$$\hat{u}_0 = \mathfrak{F}(f(x)) \tag{1.15}$$

[3]
The forward euler scheme is used here because it is fairly easy to implement. By applying the inverse discrete Fourier transform (IDFT) to $\hat{u}_a$ an approximate solution to 1.6 can be obtained. To decrease computing time, the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT) is used instead of the DFT and IDFT.

1.2. Finite Element Method

The finite element method (FEM) is a method to approximate solutions for differential equations (DE) within a certain domain Ω. This is done by discretizing the spacial domain. Assume that a DE is given by:

$$m, n \in \mathbb{N} \quad \zeta \in \Omega \subset \mathbb{R} \quad m \geq 1 \tag{1.16}$$

$$\frac{\partial^m y}{\partial \zeta^m} - g(y) = r(\zeta, t) \tag{1.17}$$

It is assumed that g is a linear function that can also contain partial derivatives of y w.r.t. time, y takes the value 0 at the boundary Γ and $y(\zeta, 0) = f(\zeta)$. An approximate solution to y is given by μ, which is expressed as a sum of basis functions contained in the set ϕ:

$$\mu(\zeta, t) = \sum_{j=1}^{N} c_j(t)\phi_j(\zeta) \tag{1.18}$$

The residual is defined as:

$$\mathfrak{r} = \frac{\partial^m \mu}{\partial \zeta^m} - g(\mu) - r(\zeta, t) \tag{1.19}$$

Furthermore the residual is required to be orthogonal to all basis functions:

$$\langle \mathfrak{r}, \phi_k \rangle = 0 \quad \forall \phi_k \in \phi \tag{1.20}$$

Since the functions in ϕ are known, it is only required to find the coefficients $c_j(t)$ in 1.18. To find those coefficients 1.20 needs to be expressed as follows:

$$\int_\Omega \frac{\partial^m \mu}{\partial \zeta^m}\phi_k \, d\zeta - \int_\Omega g(\mu)\phi_k \, d\zeta = \int_\Omega r(\zeta, t)\phi_k \, d\zeta \quad \forall \phi_k \in \phi \tag{1.21}$$

If μ is substituted with 1.18 the following is obtained:

$$\sum_{j=1}^{N}((\int_\Omega \frac{\partial^m \phi_j}{\partial \zeta^m}\phi_k \, d\zeta)c_j(t) - g((\int_\Omega \phi_k \phi_j d\zeta)c_j(t))) = \int_\Omega r(\zeta, t)\phi_k \, d\zeta \quad \forall \phi_k \in \phi \tag{1.22}$$

It is also necessary to apply divergence theorem to the first integral term taking into account that y at Γ is 0. Since ζ is one dimensional, the divergence theorem becomes integration by parts:

$$\int_\Omega \frac{\partial^m \phi_j}{\partial \zeta^m}\phi_k \, d\zeta = -\int_\Omega \frac{\partial^{m-1}\phi_j}{\partial \zeta^{m-1}}\frac{\partial \phi_k}{\partial \zeta} \, d\zeta \quad \forall \phi_k \in \phi \tag{1.23}$$

Combining 1.22 and 1.23 yields:

$$-\sum_{j=1}^{N}((\int_{\Omega}\frac{\partial^{m-1}\phi_j}{\partial\zeta^{m-1}}\frac{\partial\phi_k}{\partial\zeta}\,d\zeta)c_j(t)+g((\int_{\Omega}\phi_k\phi_j d\zeta)c_j(t)))=\int_{\Omega}r(\zeta,t)\phi_k\,d\zeta\quad\forall\phi_k\in\phi \tag{1.24}$$

This formulation leads to a system of ODEs or a system of linear equations that can be solved either analytically or numerically.

1.2.1. Solving the Heat Equation using FEM

This formulation of FEM can be applied to 1.6:

$$\Omega=\chi\quad\Gamma=\{x_0,x_n\} \tag{1.25}$$

$$y(\zeta,t)=-u(x,t)\quad g(u)=-\frac{1}{\alpha}\frac{\partial u}{\partial t} \tag{1.26}$$

$$m=2\quad r(\zeta,t)=\frac{1}{\alpha}h(x,t) \tag{1.27}$$

$$u(x,0)=f(x)\quad u(x_0,t)=0\quad u(x_n,t)=0 \tag{1.28}$$

The set of basis functions is defined as a set of piecewise linear functions with constant step size Δx:

$$\phi_j(x)=\begin{cases}(x-x_{j-1})/\Delta x), & x_{j-1}\le x<x_j\\(x_{j+1}-x)/\Delta x), & x_j\le x<x_{j+1}\\0, & \text{otherwise}\end{cases} \tag{1.29}$$

[4]
The stepsize Δx is defined by $\Delta x=\frac{x_n-x_0}{n-1}$. This results in the following system of ODEs:

$$\sum_{j=1}^{N}(\int_{\chi}\phi_j\phi_k dx)\frac{dc_j}{dt}=\alpha\sum_{j=1}^{N}(-\int_{\chi}\frac{d\phi_j}{dx}\frac{d\phi_j}{dx}dx)c_j(t)+\int_{\chi}h(x,t)\phi_k dx\quad\forall\phi_k\in\phi \tag{1.30}$$

Using matrix notation this becomes:

$$M^{N\times N},K^{N\times N} \tag{1.31}$$

$$M\dot{c}=Kc+d \tag{1.32}$$

The matrices M and K can be easily computed (Appendix A.1):

$$m_{ij}=\begin{cases}\frac{2\Delta x}{3}, & k=j\\\frac{\Delta x}{6}, & |k-j|=1\\0, & otherwise\end{cases}\qquad k_{ij}=\begin{cases}\frac{-2\alpha}{\Delta x}, & k=j\\\frac{\alpha}{\Delta x}, & |k-j|=1\\0, otherwise\end{cases} \tag{1.33}$$

However it is necessary to approximate d for each point in time using numerical integration schemes. Furthermore to solve this system of ODEs numerically an initial condition c_0 has to be known. [3]
It can be obtained using a least squares (LS) approach:

$$\sum_{j=1}^{N} \langle \phi_j, \phi_k \rangle c_j(0) = \langle f, \phi_k \rangle \quad \forall \phi_k \in \phi \tag{1.34}$$

$$Mc_0 = F \tag{1.35}$$

$$c_0 = M^{-1}F \tag{1.36}$$

[5]
Observe that by multiplying 1.32 with M^{-1} (A.2) yields a system of ODEs:

$$\dot{c} = M^{-1}Kc + M^{-1}d \tag{1.37}$$

This system of ODEs can be solved using an euler scheme:

$$c_{t+1} = \Delta t M^{-1}(Kc_t + d) + c_t \tag{1.38}$$

Force boundary conidtions

Keep in mind that vector d is time dependent and has to be recomputed for each time step. To force the boudary condition $u(x,t) = 0\ x \in \Gamma$ the first and last entry of any c_t has to be zero:

$$c_t^1 = 0 \quad c_t^n = 0 \quad \forall t \tag{1.39}$$

Therefore the Matrix M^{-1} has to be adjusted:

$$0 = \Delta t m(Kc + d) \tag{1.40}$$

$$\Rightarrow m = [0, 0, 0, \ldots, 0] \tag{1.41}$$

Here m is the first row vector of M^{-1} or the last one respectively [3]. Using 1.18 and the computed coefficients c the function $u(x,t)$ can be approximated. However this is equivalent to linear interpolation between c_n and c_{n+1} (Appendix A.3).

1.3. Singular Value Decomposition

The Singular Value Decomposition (SVD) is a matrix factorization with guaranteed existance. It can be used to obtain low rank approximations of a matrix or pseudo inverses for ill posed linear system of equations. It is also related to FT by providing a data specific set of orthogonal bases instead of a gerneric set of sines and cosines. For this paper the SVD will be used for generating low rank approximations of matrices. [6]

1.3.1. Properties

A matrix $X \in \mathbb{C}^{n \times m}$ can be decomposed in the following way:

$$X = U\Sigma V^* \tag{1.42}$$

Here $U \in \mathbb{C}^{n \times n}$ and $V \in \mathbb{C}^{m \times m}$ are unitary matrices and $\Sigma \in \mathbb{R}^{n \times m}$ is a real valued oredered diagonal matrix. The columns of U provides a set of orthonormal basis vectors for the column space of X, V contains orthonormal basis vectors for the row space of X. The matrix Σ asigns a magnitude ('importance') to the product of U and V^* [7]. Since U and V are unitary they have the following property:

$$U^*U = UU^* = I \tag{1.43}$$
$$V^*V = VV^* = I \tag{1.44}$$

[8]
In case $n \geq m$ the so called economy SVD can be used to factorize the matrix X:

$$X = \begin{bmatrix} \hat{U} & \hat{U}^\perp \end{bmatrix} \begin{bmatrix} \hat{\Sigma} \\ 0 \end{bmatrix} V^* = \hat{U}\hat{\Sigma}V^* \tag{1.45}$$

The economy SVD omits rows only containing zeros in Σ and the according columns of U. Therefore the dimensionality of $\hat{U}$ and $\hat{\Sigma}$ is less or equal to the dimensionality of U and Σ. [6]

1.3.2. Hirachy of correlations

As alaready stated, the matrix Σ assigns a magnitude to UV^*. This magnitude can be seen as the square of the variance the bases in U and V capture. Assume that X^*X and XX^* denote a correlation matrix. [**brunton_kutz_2019**] A correlation matrix is a matrix that stores correlation coefficiants between multible measurements. If X has the following properties X^*X and XX^* are correlation matrices:

1.) The column vectors of X have to be zero mean:

$$X = \left[x_1, \cdots, x_m\right] \tag{1.46}$$

$$\frac{1}{n}\sum_{j=1}^{n} x_{ij} = \mu_i = 0 \quad \forall 1 \leq i \leq m \tag{1.47}$$

2.) The column vectors of X have to be normalized:

$$(\sum_{j=1}^{n} x_{ij}^2)^{\frac{1}{2}} = \bar{x_i} = 1 \quad \forall 1 \leq i \leq m \tag{1.48}$$

The correlation coefficiant between two column vectors of X is calculated as follows:

$$corr(x_i, x_{i'}) \frac{(\sum_{j=1}^{n} x_{ij} - \bar{x_i})(\sum_{j=1}^{n} x_{i'j} - \bar{x_{i'}})}{(\sum_{j=1}^{n}(x_{ij} - \bar{x_i})^2)^{\frac{1}{2}}(\sum_{j=1}^{n}(x_{i'j} - \bar{x_{i'}})^2)^{\frac{1}{2}}} \tag{1.49}$$

[9]
Since all coloumn vectors have zero mean and are normalized this becomes:

$$corr(x_i, x_{i'}) = cov(x_i, x_{i'}) = x_i^* x_{i'} \tag{1.50}$$

[10]
This resembles the entries of XX^* and X^*X. A vector e that maximizes the variance of the projecetions of x_i onto e with the restriction $\|e\| = 1$ are the eigenvectors of the according correlation matrix. The eigenvalue of e denoted as λ is eqiuvalent to the variance of x_i projected onto e [11].
Since X can be deconstructed using the SVD, XX^* and X^*X are equal to:

$$XX^* = U\begin{bmatrix}\hat{\Sigma}\\ 0\end{bmatrix}V^*V\begin{bmatrix}\hat{\Sigma} & 0\end{bmatrix}U^* = U\begin{bmatrix}\hat{\Sigma}^2 & 0\\ 0 & 0\end{bmatrix}U^* \tag{1.51}$$

$$X^*X = V\begin{bmatrix}\hat{\Sigma} & 0\end{bmatrix}U^*U\begin{bmatrix}\hat{\Sigma}\\ 0\end{bmatrix}V^* = V\hat{\Sigma}^2V^* \tag{1.52}$$

By multiplying U and V respectivley on the right side 1.51 and 1.52 become:

$$XX^*U = U\begin{bmatrix}\hat{\Sigma}^2 & 0\\ 0 & 0\end{bmatrix} \tag{1.53}$$

$$X^*XV = V\hat{\Sigma}^2 \tag{1.54}$$

This shows that V contains the eigenvectors of the row-wise correlation matrix and U contains the eigenvectors of the column-wise correlation matrix. The matrix Σ contains the roots of the according eigenvalues and is thereby related to the variance. By ordering U, Σ and V by

the entries of Σ in a descending order he first row of U and V contain the most important basis vectors. [6]

1.3.3. Low-rank approximation

A usefull property of the SVD is that it can be used to find an hierachy of rank-r approximation for a given matrix X. An matrix $\tilde{X}$ that approximates X is obtained by:

$$\tilde{X} = arg\,min\|X - \tilde{X}\|_F = \tilde{U}\tilde{\Sigma}\tilde{V}^* \tag{1.55}$$

$$s.t. rank(\tilde{X}) = r \tag{1.56}$$

Here $\tilde{U}$ and $\tilde{V}$ denote matrices obtained takeing the first r columns of U and V. The matrix $\tilde{\Sigma}$ is a $r \times r$ sub-block of Σ. This is alsow known as the Eckard-Young theorem. The variance captured by $\tilde{X}$ can be calculated in the following way:

$$cumvar_r(\Sigma) = \frac{\sum_{i=1}^{r} \sigma_i}{trace(\Sigma)} \tag{1.57}$$

$$var(\Sigma) = \frac{diag(\Sigma)}{trace(\Sigma)} \tag{1.58}$$

[6]

1.3.4. Example low-rank approximation

As an example suppose there is a matrix $X \in \mathbb{Z}^{4\times4}$:

$$X = \begin{bmatrix} 3 & 1 & 5 & 5 \\ 4 & -4 & 5 & 0 \\ -4 & -2 & -4 & 3 \\ 5 & 1 & 5 & -4 \end{bmatrix} \tag{1.59}$$

By computing the SVD the matrices U, Σ and V are obtained. Now Σ can be used to calculate the cummulative variance for an rank-r approximation and the variance captured by each basis vector of U and V.

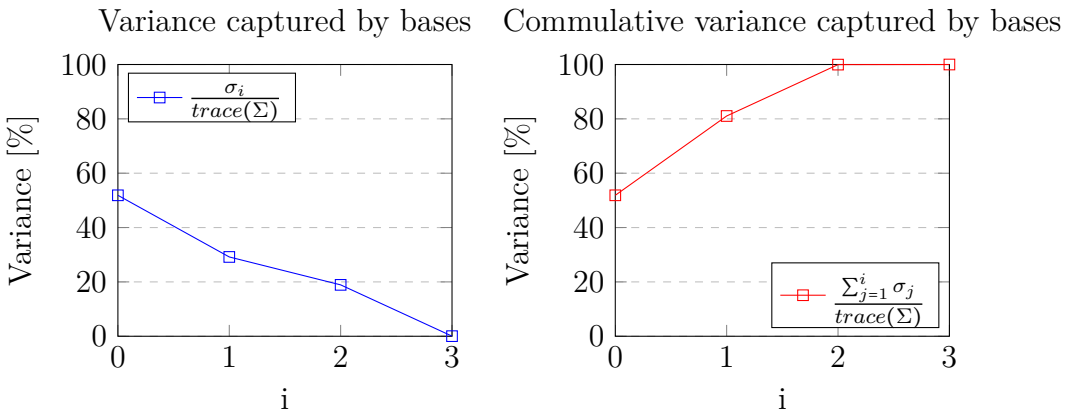

Figure 1.1.: Variance and commulative variance captured by each column vectors of U and V

On 1.3.4 the commulative variance and the variance of each basis vectors is plotted. Here X can be apprixomated using the first three leading basis vectors. This approximation captures already more than 99% of the variance. The resulting matrix $\tilde{X}_3$ looks as follows:

$$\tilde{X}_3 = \begin{bmatrix} 4.73 & -0.76 & 5.62 \\ 1.31 & -1.37 & 1.62 \\ -5.10 & -0.60 & 2.34 \end{bmatrix} \tag{1.60}$$

1.4. Fundamentals of Control Theory

1.4.1. Problems in Control Theory

There are mainly three different problems in control theory. To analyse a system it has to be identified first. A mathematical model has to be found that relates the input of that system to it's output. After a model has been found, it is helpfull to simulate the system to find out which input results in which output. Simulations have to fit the system and offer the advantage over real experiments of reducing cost and risks attached to them. The last of the three major problems is the control problem. Now that the underlying model is known and the system can be simulated, a controler for that system can be designed. Here the controler is a system that feeds some input to the system that is to be controlled to generate some desired output. [12] For this paper the second problem is the most important one, since the goal of this paper is to compare some methods to make simulations of a system more efficient.

1.4.2. LTI Systems

A linear and time invariant (LTI) system is a kind of system that fulfills the following conditions:

Homogenity: For a system to be a LTI system, the condition that the output of a system $y(t)$ relates to the input $u(t)$ by a linear operator h:

$$y(t) = h(u(t)) \tag{1.61}$$
$$cy(t) = h(cu(t)) \tag{1.62}$$

This means that if the input u changes in magnitude by a constant factor of c the output y also changes by the factor of c.

Superposition: The second requirement is superpostion:

$$y_1(t) + y_2(t) = h(u_1(t) + u_2(t)) \tag{1.63}$$

The sum of two outputs y_1 and y_2 has to equal the output of the system with the sum of the inputs u_1 and u_2 as input.

Time Invariance: A system that is time invariant has the property that if the input u is shifted in time by some constant τ the resulting output is also shifted in time by the same constant:

$$y(t-\tau) = h(u(t-\tau)) \tag{1.64}$$

LTI systems are important because they can be used to approximate non LTI systems over some region. This is usefull since LTI systems are well understood. [13]

1.4.3. State Space representation

A LTI system can be expressed as a system of ODEs that relates the input u to its output y. The vector x denotes the states of a system.

$$\dot{x} = Ax + Bu \quad x(t_0) = x_0 \tag{1.65}$$

$$y = Cx + Du \tag{1.66}$$

The matrices $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{q \times n}$ and $D \in \mathbb{R}^{q \times n}$ are constant matrices. [14]

1.4.4. Transfer Function

1.4.5. Controallability and Observability

2. Model Order Reduction

2.1. Introduction

Model Order Reduction (MOR) is a technique to reduce the computational effort of simulate a system using mathematical models. This is done by using two different approaches. The first one is using a numerical approach and the second approach is based on system theory. The first method discussed is called Proper Orthogonal Decomposition. After that Balanced and Modal Truncation will be discussed. The last method discussed will be Hankel-norm approximation. [15].

2.2. Proper Orthogonal Decomposition

The proper orthogonal decomposition (POD) is a method for model order reduction. The reduction in computational effort is done by approximating a solution to a PDE using an orthogonal expansion. The basis functions are obtained by decomposing a set of solutions to othe PDE using the SVD.

2.2.1. Orthogonal Expansion

A function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ can be represented by the series:

$$f(x,t) = \sum_{i=1}^{\infty} a_i(t)\phi_i(x) \tag{2.1}$$

$$x, t \in \mathbb{R} \tag{2.2}$$

Here all $\phi(x)$ are orthorgonal basis functions.

$$\langle \phi_i, \phi_j \rangle = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{else} \end{cases} \tag{2.3}$$

By using a finite sum instead of the entire series f can be approximated:

$$\tilde{f}(x,t) = \sum_{i=1}^{n} a_i(t)\phi_i(x) \tag{2.4}$$

Since all ϕ are known, 2.4 has to be solved for the set $\{a_0, ..., a_n\}$. In case f is known, the solutions contained in this set can be calculated as:

$$a_i(t) = \frac{\langle \phi_i, f \rangle}{\langle \phi_i, \phi_i \rangle} \tag{2.5}$$

[16]

2.2.2. Proper Orthogonal Decomposition for PDEs

Since a PDEs are equations in terms of partial derivatives, the notation $\mathscr{P}(\partial x)u$ is introduced, which denotes a differential operator in terms of the spacial variables x for a function $u(x, t, p.$ The vector p contains some parameters. This is done to provide a more abstract way to denote PDEs:

$$\frac{\partial u}{\partial t} = \mathscr{P}(\partial x)u \tag{2.6}$$

[17] Solving for u is often dificult to imposible. A method that is often used to solve PDEs is called seperation of variables. This spereation of variables assumes, that the underlying solution $u(x,t)$ can be expressed by 2.2, to solve for $a_k, 0 \leq k \leq n$. Since it is not practical to compute an infite series, u only gets appriximated by using 2.4 instead:

$$\sum_{i=1}^{n} \frac{\partial a_i}{\partial t}\phi_i = \sum_{i=1}^{n} \mathscr{P}(\partial x)\phi_i a_i \tag{2.7}$$

By discretizing the spacial dimension got along x 2.7 can be expressed using marix notation:

$$\Phi = \left[\phi_0, ..., \phi_n\right] \tag{2.8}$$

$$\Phi \frac{d}{dt} a = \mathscr{P}(\partial x)\Phi a \tag{2.9}$$

Since the solution u is unknown 2.5 cannot be computed. However the fact, that all ϕ are orthogonal to each oter, can be used to solve for all a_k. It can be done by computing the inner product of 2.7 with all basis funktions:

$$\langle \sum_{i=1}^{n} \frac{\partial a_i}{\partial t}\phi_i, \phi_k \rangle = \langle \sum_{i=1}^{n} \mathscr{P}(\partial x)\phi_i a_i, \phi_k \rangle \quad \forall 0 \leq k \leq n \tag{2.10}$$

This resembles the Galerking projection. In matrix notation it can be expressed:

$$\Phi^* \Phi \frac{d}{dt} a = \Phi^* \mathscr{P}(\partial x)\Phi a \tag{2.11}$$

By considering 2.3 the equiation 2.10 can be expressed as a system of ODEs which can be solved:

$$I \frac{d}{dt} a = \Phi^* \mathscr{P}(\partial x)\Phi a \tag{2.12}$$

After the vector of coefficiants a for each time step has been computed, the solution can be assembled:

$$u(x,t) \approx \Phi(x)a(t) \tag{2.13}$$

[18]

2.2.3. Choosing basis vectors

As discussed in the previous section, a set of basis vectors can be used to generate approximate solutions to PDEs. However it was not discussed how those basis functions are chosen. For POD to work, a so called snapshot matrix X has to be available. This snapshot matrix stores

a set of solutions where x_k is the solution for a PDE at time step $k\Delta t$:

$$X = \left[x_1, ..., x_m\right] \tag{2.14}$$

The solutions can be obtained by conducting an expirment on a physical system that is described by the PDE or by simulating the evolution of that PDE. In this paper the solution is obtained using FEM 1.2. SVD is used to decompose the snapshot matrix X. Since the columns of the snapshot matrix contain spacial information at a given point in time and the basis vectors are supposed to encode the spacial information of a solution u the r most dominant left singular values have to be extracted:

$$\tilde{X} = \tilde{U}\tilde{\Sigma}\tilde{V}^* \tag{2.15}$$

$$\Phi = [u_1, ..., u_r] \tag{2.16}$$

Here the FEM solution is obtained by discretizing the PDE into a large number(n) of spacial nodes. This results in a high dimensional system of ODEs. Since Φ contains only r column ($r << n$) vectors a reduction in order can be achieved. However the benefits of this reduced order model are only relevant after the PDE has been solved once using a high dimensional system of ODEs. [18]

2.3. Balanced Truncation

3. Implementation

The implementation of the discussed methods for solving the heat equation and for model order reduction was done in Matlab. Matlab was chosen as the programming language because it natively features matrix multiplication which finds heavy use in the previously mentioned methods. The second reason for this selection is that there exist ToolBoxes that already implement certain model order reduction methods such as MORLAB [19] or MOR toolbox [20]. The following figure shows the class diagramm of the implementation:

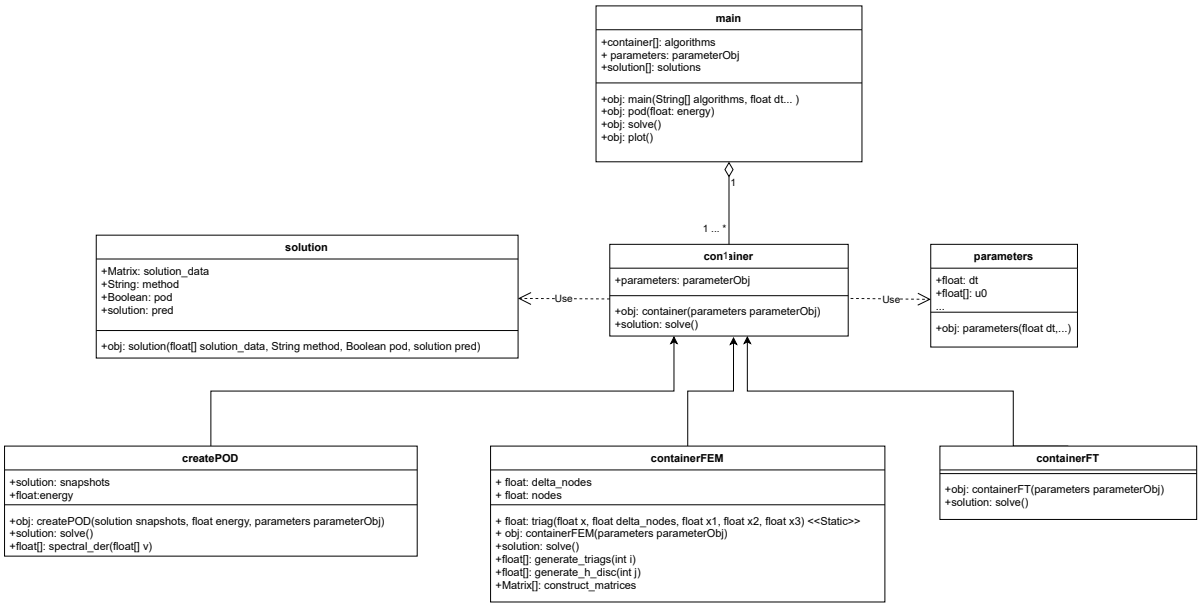

Figure 3.1.: Class diagramm of MOR and FEM implementation

3.1. Class main

The class main is responsible for generating the finite element solution, model order reduction steps and plotting the results. The process can be seen in the following flow chart:

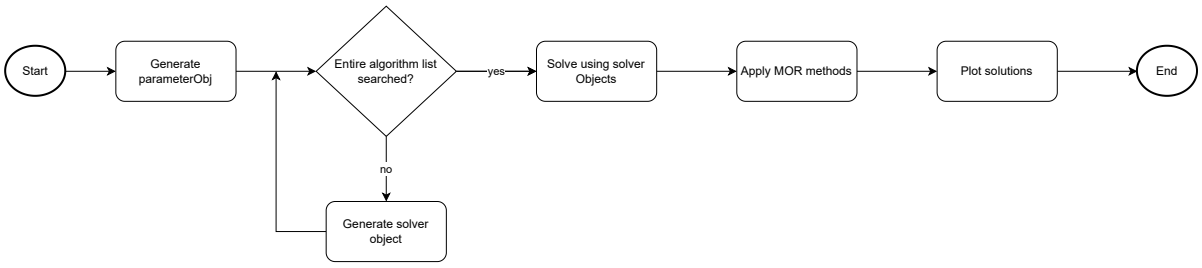

Figure 3.2.: Flow chart of main class

The first step is to generate a parameter object. The parameter object stores all parameters in order to increase the transparancy and robustness of the programm. The second step is to iterate the array of stated algorithms to solve the heat equation. The options are to solve the heat equation using finite element method 1.2 or using fourier transform 1.1. After all solver objects have been generated, the according solutions are being computed. After that, the MOR methods are displayed. The final step is to display the solutions.

3.2. Class containerFEM

The class containerFEM is responsible for generating a solution using finite element method. FEM is implemented in the following way:

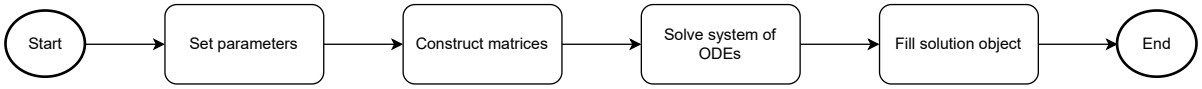

Figure 3.3.: Flow chart for FEM class

The first step is to set the parameters. After that the matrices discussed in 1.2 are being constructed. The next step is to solve the resulting system of ODEs and pass the solution to a solution object.

3.2.1. Construct Matrices

As defined in 1.33 the matrices K and M have to be constructed. Also to compute the initial condition given by u_0 F has to be knows 1.35. This is done by the following method:

Algorithm 1 Construct matrices K, M and F

$ii \leftarrow \frac{2}{3}\Delta nodes$
$ij \leftarrow \frac{1}{6}\Delta nodes$
$F \leftarrow zeros(nodes, 1)$
$K \leftarrow zeros(nodes)$
$M \leftarrow zeros(nodes)$
for $i = 1$ **to** $nodes$ **do**
    $F(i) \leftarrow trapz(\phi_i \cdot u_0)$
end for
for $i = 1$ **to** $nodes$ **do**
    $K(i,i) \leftarrow \frac{-2}{\Delta nodes}$
    $M(i,i) \leftarrow ii$
    if $i - 1 > 1$ **then**
        $K(i,i-1) \leftarrow \frac{1}{\Delta nodes}$
        $M(i,i-1) \leftarrow ij$
    end if
    if $i + 1 < nodes + 1$ **then**
        $K(i,i+1) \leftarrow \frac{1}{\Delta nodes}$
        $M(i,i+1) \leftarrow ij$
    end if
end for
return$[F, K, M]$

3.2.2. Compute vector d(t)

As discussed in 1.2 vectord d has to be computed for each time step:

Algorithm 2 Construct vector d

$d \leftarrow zeros(nodes, 1)$
for $i = 1$ **to** $nodes$ **do**
    $d(i) \leftarrow trapz(\phi_i \cdot h(x, t_0))$
end for
return d

The entries of vector d become the integral of the product of a basisfunction and h evaluated at timestep t_0. This timestep is a argument of that method.

3.2.3. Solve System of ODEs

The most important step in the process of generating a solution is to solve the system of ordinary differential equations that FEM yields:

Algorithm 3 Solve system of ODEs using euler's scheme

1: $[F, K, M] \leftarrow construct\_matrices()$
2: $C \leftarrow zeros(nodes, n\_time\_steps)$
3: $c_0 \leftarrow M^{-1}F$
4: $C(:, 1) \leftarrow c_0$
5: $M(1, :) \leftarrow [0, \cdots, 0]$
6: $M(end, :) \leftarrow [0, \cdots, 0]$
7: $N \leftarrow M^{-1}K$
8: **for** $t = 2$ **to** n_time_steps **do**
9: $\quad d \leftarrow generate\_h\_disc(t)$
10: $\quad c_n \leftarrow \Delta t N c_0 + M^{-1}h + c_0$
11: $\quad c_0 \leftarrow c_n$
12: $\quad C(:, t) \leftarrow c_n$
13: **end for**
14: $S \leftarrow []$
15: **for** $t = 1$ **to** n_time_steps **do**
16: $\quad c \leftarrow C(:, t)$
17: $\quad interpol \leftarrow interp1(linspace(0,\ L,\ nodes), c,\ X)$
18: $\quad S(:, t) \leftarrow interpol$
19: **end for**
20: **return** $solution(S,\ "FEM",\ 0,\ 0)$

In the first line the matrices F, K and M are retrieved. After that the initial vector of coefficiants c_0 is computed using LS 1.36 in line three. The next two following lines force the boundary conditions as stated in 1.41. In the lines 8 to 13 1.38 is implemented. In the last step the coefficiants are interpolated in spatial direction to fit the given domain X and stored in a solution object.

Literaturverzeichnis

[1] Gustafsson, B. "Heat Conduction". In: *Fundamentals of Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 255–262. URL: https://doi.org/10.1007/978-3-642-19495-5_16.

[2] Papula, L. "Gewöhnliche Differentialgleichungen". In: *Mathematik für Ingenieure und Naturwissenschaftler Band 2: Ein Lehr- und Arbeitsbuch für das Grundstudium*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, pp. 343–541. URL: https://doi.org/10.1007/978-3-658-07790-7_4.

[3] Gustafsson, B. "Finite Element Methods". In: *Fundamentals of Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 173–192. URL: https://doi.org/10.1007/978-3-642-19495-5_11.

[4] Gustafsson, B. "Polynomial Expansions". In: *Fundamentals of Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 117–123. URL: https://doi.org/10.1007/978-3-642-19495-5_7.

[5] Gustafsson, B. "Least Square Problems". In: *Fundamentals of Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 125–133. URL: https://doi.org/10.1007/978-3-642-19495-5_8.

[6] Brunton, S. L./ Kutz, J. N. "Singular Value Decomposition (SVD)". In: *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019, pp. 3–46.

[7] Brunton, S. L./ Kutz, J. N. "Fourier and Wavelet Transforms". In: *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019, pp. 47–83.

[8] Szabo, F. E. "U". In: *The Linear Algebra Survival Guide*. Ed. by Szabo, F. E. Boston: Academic Press, 2015, pp. 385–392. URL: https://www.sciencedirect.com/science/article/pii/B978012409520550028X.

[9] Naoko, S. *Geometric Interpretation of the Correlation between Two Variables*. 2018. URL: https://medium.com/@ns2586/geometric-interpretation-of-the-correlation-between-two-variables-4011fb3ea18e (visited on 03/01/2023).

[10] Oliver, K. *Math 19b: Linear Algebra with Probability*. 2011. URL: https://people.math.harvard.edu/~knill/teaching/math19b_2011/handouts/lecture12.pdf (visited on 03/01/2023).

[11] Victor, L. *IAML: Dimensionality Reduc6on*. 2011. URL: https://www.inf.ed.ac.uk/teaching/courses/iaml/2011/slides/pca.pdf (visited on 03/01/2023).

[12] Douglas, B. "The Fundamentals of Control Theory". In: Academia, 2019, pp. 8–15.

[13] Douglas, B. "The Fundamentals of Control Theory". In: Academia, 2019, pp. 42–47.

[14] Benner, P. et al. "1 Model order reduction: basic concepts and notation". In: *Volume 1 System- and Data-Driven Methods and Algorithms*. Ed. by Benner, P. et al. Berlin, Boston: De Gruyter, 2021, pp. 1–14. URL: https://doi.org/10.1515/9783110498967-001.

[15] *Volume 1 System- and Data-Driven Methods and Algorithms*. Berlin, Boston: De Gruyter, 2021. URL: https://doi.org/10.1515/9783110498967.

[16] Gustafsson, B. "Polynomial Expansions". In: *Fundamentals of Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 117–123. URL: https://doi.org/10.1007/978-3-642-19495-5_7.

[17] Gustafsson, B. "Numerical Methods for Differential Equations". In: *Fundamentals of Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 137–144. URL: https://doi.org/10.1007/978-3-642-19495-5_9.

[18] Brunton, S. L./ Kutz, J. N. "Reduced Order Models (ROMs)". In: *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019, pp. 375–402.

[19] Benner, P./ Werner, S. W. R. *Morlab*. URL: https://www.mpi-magdeburg.mpg.de/projects/morlab (visited on 02/27/2023).

[20] Systems, P. D. *MOR toolbox*. URL: https://mordigitalsystems.fr/en/products.html (visited on 02/27/2023).

[21] D'Erchie, M. e. a. "Analysis Eins". In: *Mathe für Nicht-Freaks*. Serlo Education e. V., 2017, pp. 111–113. URL: https://mathe-builds.serlo.org/analysis1_final/analysis1_final.pdf.

[22] Alexandre M. Bayen, T. S. “An Introduction to MATLAB® Programming and Numerical Methods for Engineers”. In: Science Direct, 2015, pp. 211–214. URL: https://www.sciencedirect.com/book/9780124202283/an-introduction-to-matlab-programming-and-numerical-methods-for-engineers.

A. Appendix

A.1. Deriving matrices for FEM using piecewise linear functions

The so called triangle function is defined as follows:

$$\phi_j(x) = \begin{cases} (x - x_{j-1})/\Delta x), & x_{j-1} \leq x < x_j \\ (x_{j+1} - x)/\Delta x), & x_j \leq x < x_{j+1} \\ 0, \quad \text{otherwise} \end{cases} \tag{A.1}$$

[4] The following integrals have to be evaluated:

$$\int_\chi \phi_j \phi_k dx \quad \forall \phi_k \in \phi \tag{A.2}$$

$$-\int_\chi \frac{d\phi_j}{dx}\frac{d\phi_k}{dx} dx \quad \forall \phi_k \in \phi \tag{A.3}$$

With $\chi \subset \mathbb{R}$. Note that the product of two functions ϕ_j and ϕ_k and their derivatives is only under two conditions not zero:

1. $k = j$

Considering this case the integral A.2 becomes:

$$\int_{x_{j-1}}^{x_j} \phi_j^2 dx + \int_{x_j}^{x_{j+1}} \phi_j^2 dx \tag{A.4}$$

Because of symmetry only one of the above integrals have to computed:

$$2\int_{x_{j-1}}^{x_j} \phi_j^2 dx \tag{A.5}$$

$$= \frac{2}{\Delta x^2}\int_{x_{j-1}}^{x_j} (x - x_{j-1})^2 dx \tag{A.6}$$

$$\frac{2}{3\Delta x^2}\left[(x - x_{j-1})^3\right]_{x_{j-1}}^{x_j} = \frac{2}{3\Delta x^2}\Delta x^3 = \frac{2}{3}\Delta x \tag{A.7}$$

Integral A.3 for $i = j$ taking symmetry into account becomes:

$$-\int_{x_{j-1}}^{x_{j+1}} (\frac{d\phi_j}{dx})^2 dx = -\frac{1}{\Delta x^2}\int_{x_{j-1}}^{x_{j+1}} 1dx \tag{A.8}$$

$$= -\frac{1}{\Delta x^2}\left[x\right]_{x_{j-1}}^{x_{j+1}} = -\frac{2}{\Delta x} \tag{A.9}$$

2. $|j-k| = 1$ A.2 becomes:

$$\frac{1}{\Delta x^2}\int_{x_j}^{x_{j+1}} (x - x_j)(x_{j+1} - x)dx \tag{A.10}$$

$$= \left[\frac{1}{2}x^2 x_{j+1} - \frac{1}{3}x^3 - x x_{j+1} x_j + \frac{1}{2}x^2 x_j\right]_{x_j}^{x_{x_j+1}} = \frac{1}{6\Delta x^2}\Delta x^3 = \frac{1}{6}\Delta x \tag{A.11}$$

Finally A.3 has to be evaluated for this condition:

$$-\int_{x_j}^{x_{j+1}} \frac{d\phi_j}{dx}\frac{d\phi_{j+1}}{dx}dx = \frac{1}{\Delta x^2}\int_{x_j}^{x_{j+1}} 1dx = \frac{1}{\Delta x^2}\left[x\right]_{x_j}^{x_{j+1}} = \frac{1}{\Delta x} \tag{A.12}$$

A.2. Proof that matrix M is invertible

Let M_n be a matrix with $M_n \in \mathbb{R}^{n\times n}$ given by:

$$m_{ij} = \begin{cases} a, & k = j \\ b, & |k-j| = 1 \\ 0, & otherwise \end{cases} \tag{A.13}$$

It's determinant is given by the Laplace expansion:

$$det(M_n) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} N_{ij} \quad \forall i \tag{A.14}$$

N_{ij} is the determinant of the matrix M' that is obtained by removing the i^{th} row and j^{th} column of M_n. This expression can be simplified using the definition of m_{ij}:

$$det(M_n) = aN_{11} - bN_{12} \tag{A.15}$$

N_{11} is equivalent to $det(M_{n-1})$, since the indices of rows and columns of M' are in consecutive order and M' is a $n-1 \times n-1$ matrix :

$$N_{11} = det(M') \tag{A.16}$$

$$M' = \begin{bmatrix} m_{22} & \dots & m_{2n} \\ \vdots & \ddots & \vdots \\ m_{n2} & \dots & m_{nn} \end{bmatrix} \tag{A.17}$$

N_{12} can be obtained by calculating the determinant of M' using the Laplace expansion:

$$M' = \begin{bmatrix} m_{21} & m_{23} & \dots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n3} & \dots & m_{nn} \end{bmatrix} \tag{A.18}$$

$$det(M') = m_{21} \cdot det(M'') \tag{A.19}$$

$$M'' = \begin{bmatrix} m_{33} & \dots & m_{3n} \\ \vdots & \ddots & \vdots \\ m_{n3} & \dots & m_{nn} \end{bmatrix} \tag{A.20}$$

In A.19 only the stated term has to evaluated since all entries of the first column of the second submatrix are zero. Therefore the determinant is zero. The row and column indices of M'' are in consecutive order and it is a $n-2 \times n-2$ matrix. Therefore M'' is equivalent to M_{n-2}. A.15 becomes:

$$det(M_n) = a \cdot det(M_{n-1}) - b^2 \cdot det(M_{n-2}) \tag{A.21}$$

Furthermore this implies $det(M_0) = 1$:

$$det(M_2) = a^2 - b^2 = a \cdot det(M_1) - b^2 \cdot 1 \tag{A.22}$$

$$\Rightarrow det(M_0) = 1 \tag{A.23}$$

Using the definition of 1.33 and 1.2.1 this can be seen as the following sequence:

$$a_0 = 1,\ a_1 = \frac{2\Delta x}{3} \tag{A.24}$$

$$a_{n+1} = \frac{2\Delta x}{3} \cdot a_n - \frac{\Delta x^2}{36} \cdot a_{n-1} \tag{A.25}$$

As described here [21] a recursive sequence converges if it is monotone and has a limit. A proof by induction shows that this sequence is monotone for $n \geq 1$.

Base case:

$$a_2 = (\frac{2\Delta x}{3})^2 - \frac{\Delta x^2}{36} = \Delta x^2(\frac{4}{9} - \frac{1}{36}) < \frac{2\Delta x}{3} = a_1 \tag{A.26}$$

Induction step: Assuming that $a_k < a_{k-1}$ holds, $a_{k+1} < a_k$ also holds:

$$a_{k+1} = \frac{2\Delta x}{3} \cdot a_k - \frac{\Delta x^2}{36} \cdot a_{k-1} < \frac{2\Delta x}{3} \cdot a_{k-1} - \frac{\Delta x^2}{36} \cdot a_{k-2} = a_k \tag{A.27}$$

The limit of this sequence is as follow:

$$\alpha = \lim_{n\to\infty} a_{n+1} = \lim_{n\to\infty} \Delta x \cdot \frac{2}{3} \cdot \lim_{n\to\infty} a_n - \lim_{n\to\infty} \Delta x^2 \cdot \frac{1}{36} \cdot \lim_{n\to\infty} a_{n-1} = 0 \cdot \alpha - 0 \cdot \alpha = 0 \tag{A.28}$$

Since this series is monotone and converges to zero as n goes to infinity, there is no $n \in \mathbb{N}$ for which $a_n = 0$. Therefore the determinant of the matrix M defined in 1.33 is not zero and M is invertible.

A.3. Equivalence of picewise linear polynomials and linear interpolation

A picewise linear polynomial in the form of:

$$u(x,t) = \sum_{j=1}^{N} c_j(t)\phi_j(x) \tag{A.29}$$

With ϕ_j being defined as A.1 and $c_j : \mathbb{R} \to \mathbb{R}$ is equivalent to linear interpolation with respect to x:

$$\hat{u}(x,t) = u_j + \frac{(u_{j+1} - u_j)(x - x_j)}{x_{j+1} - x_j} \tag{A.30}$$

for $x_j < x < x_{j+1}$ [22]. This can be shown by evaluating $u(x,t)$ between two neigbouring ϕ and $x_j < x < x_{j+1}$:

$$u(x,t) = \phi_j(x)c_j(t) + \phi_{j+1}(x)c_{j+1}(t) \tag{A.31}$$

$$= \frac{x_{j+1} - x}{\Delta x}c_j(t) + \frac{x - x_j}{\Delta x}c_{j+1}(t) \tag{A.32}$$

$$= \frac{x_{j+1} - x}{\Delta x}u_j + \frac{x - x_j}{\Delta x}u_{j+1} \tag{A.33}$$

$$= \frac{(x_{j+1}u_j - xu_j) + (xu_{j+1} - x_ju_{j+1})}{\Delta x} \tag{A.34}$$

$$= \frac{(u_{j+1} - u_j)x + x_{j+1}u_j - x_ju_{j+1}}{\Delta x} \tag{A.35}$$

$$= \frac{(u_{j+1} - u_j)x + x_ju_j + \Delta xu_j - x_ju_{j+1}}{\Delta x} \tag{A.36}$$

$$= u_j + \frac{(u_{j+1} - u_j)x - x_j(u_{j+1} - u_j)}{\Delta x} \tag{A.37}$$

$$= u_j + \frac{(u_{j+1} - u_j)(x - x_j)}{x_{j+1} - x_j} \tag{A.38}$$